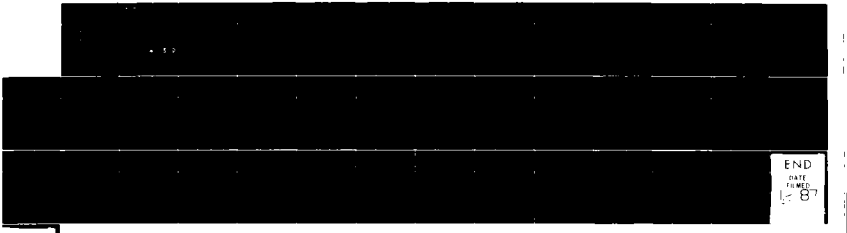
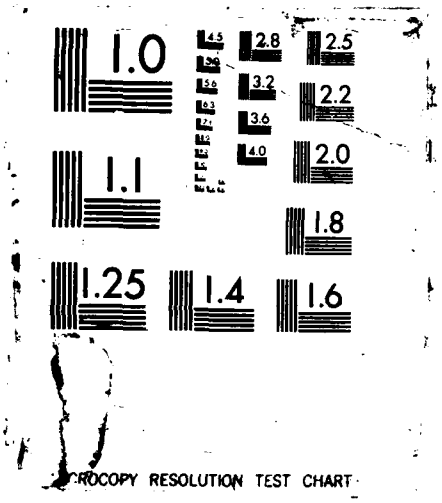
AD-A185 475 ON REAL-TIME SYSTEMS USING LOCAL AREA NETWORKS(U) 1/1
MARYLAND UNIV COLLEGE PARK INST FOR ADVANCED COMPUTER
STUDIES S LEVI ET AL. JUL 87 UMIACS-TR-87-35
UNCLASSIFIED N00014-87-0134 F G 12/7 NL



END
DATE FILMED
12-87

END
DATE FILMED
12-87


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
ROCOPY RESOLUTION TEST CHART

(12)

UMIACS-TR-87-35
CS-TR-1892

July, 1987

On Real-Time Systems Using Local Area Networks*

Shem-Tov Levi
Department of Computer Science

Satish K. Tripathi†
Department of Computer Science and
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

(12)

A

UMIACS-TR-87-35
CS-TR-1892

July, 1987

On Real-Time Systems Using Local Area Networks*

Shem-Tov Levi
Department of Computer Science

Satish K. Tripathi†
Department of Computer Science and
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

ABSTRACT

The design and the implementation of a real-time communication network has to meet goals of fault-free and timely delivery of messages, as defined by the applications served by this network. The time constraints of the applications are projected as stringent constraints on the local knowledge of time. However, the local knowledge of time in a distributed system requires the use of the communication network. The accuracy and coherence of this knowledge depend on the way clocks are synchronized. Therefore, the performance of clock synchronization algorithms due to the communication network, is a good criterion for analyzing the adequacy of a particular network architecture to real-time application.

In this paper, we examine the adequacy of LAN architecture for real-time applications through a delay model of a typical LAN. The model is applied to various clock synchronization algorithms, and error estimates for these algorithms are derived. An efficient environment, in which the uncertainties in message communication elapsed time is significantly reduced, is then described for a layered protocol architecture. In addition, it is shown that predictable communication elapsed time can be compensated for in synchronization algorithms, significantly reducing the errors of these algorithms. As an example of hierarchical protocols for this environment, we have chosen to use TCP, IP and ICMP.

* This work is supported in part by contracts N00014-87-K-0124 and N00014-87-K-0463 from the Office of Naval Research to the Department of Computer Science, University of Maryland.

† The authors thank John Zahorjan for his constructive criticism on the initial version of this paper.

Contents

1 Introduction

1.1 Scope

The use of distributed systems has widely spread in applications that need high fault tolerance. It is the relatively simple implementation of independent redundancy that makes a distributed architecture attractive. Yet, distribution of the processing resources raises many issues concerning compatibility of different processor types. These compatibility issues necessitate the use of standardized "conversation" procedures, commonly called *protocols*. A very popular distributed architecture is the *network architecture*. In this architecture no transparency of resources is imposed on the nodes of the network: each node accesses resources using their physical addresses. Locally, each node may be implemented in a different architecture, creating a high degree of heterogeneity. The only common restriction on the members of the network concerns the use of the proper communication protocol.

A difficult and fundamental task in a real-time system is the management of a distributed set of real-time clocks. This set of clocks feed the application programs with the knowledge of real-time. In addition to the restriction to maintain this knowledge locally within a strict tolerance, difference between clocks should be strictly bounded as well. These restrictions are met in a variety of clock synchronization algorithms. Some examples are given in the appendix. The accuracy of each of the algorithms depends on timing parameters of the communication network. Only a network whose timing parameters satisfy the requirements imposed by the time constraints of each of its applications may be adequate for an implementation of a real-time system.

In this paper, we investigate the adequacy of Local Area Networks (LANs) for real-time synchronization. The paper starts with introducing the real-time constraints and the clock systems that feed the time to real-time systems. A model for real-time system based on LAN communication is presented in section 2. Section 3 presents issues and algorithms for synchronization in the context of the model. In section 4 we consider TCP/IP ([15,13]) and ICMP ([14]) environment to examine the applicability of the model.

1.2 Time Constraints

A real-time system can be described as a set of time constraints. Informally speaking, a time constraint is a requirement to start executing a particular executable object after a condition is satisfied and to complete the execution before a deadline has passed. The execution time of the object is assumed to be given, and the constraint is extended to a periodic execution of the object. The formal definition of a time constraint is based on the

local knowledge of current real-time $C_i(t) = t + \Delta_i(t)$, where $C_i(t)$ is a mapping function from real time to clock "i" time (see section 3.1) and $\Delta_i(t)$ is the incorrectness of this clock. This knowledge is expressed using state predicates like

$$Taft_i(C_0)$$

which is *true* for $C_i(t) \geq C_0$ and *false* otherwise, and

$$Tbef_i(C_n)$$

which is *true* for $C_i(t) \leq C_n$ and *false* otherwise.

A time constraint is formally defined as the quintuple

$$< Id, Taft(condition_1), c_{Id}, f_{Id}, Tbef(condition_2) >$$

where:

Id is the name of the executable object (process),

$Taft(condition_1)$ states the condition after which execution should begin (Simple *true* stands for "as soon as possible"),

c_{Id} is the computation time of object Id [1],

f_{Id} is the frequency at which the computation should be carried out, in case this is a periodic process. In case of a spontaneous (sporadic) process, this is the maximal frequency expected ($f_{Id} = 0$ stands for a single occurence of an execution),

$Tbef(condition_2)$ states the deadline d_{Id} which should be met. $condition_2 \equiv C_{Id}(t) = \infty$ stands for an "off line" computation.

The resources of a distributed real-time system are allocated to execute the set of time constraints. The resource allocation process and the various schedulers require a priori knowledge about the execution times in a particular environment. This requirement implies a priori knowledge of the communication timing parameters in these optional environments. Given a particular static allocation, the system can be viewed as a network. Each processing node executes its own allocated tasks, each represented as a set of time constraints, and communicate with other sites using a communication protocol. The protocol is used to exchange time information in addition to other functional data.

In order to use LANs for real-time systems we have to study LAN models and their clock synchronization algorithms (see Appendix A) to show that a reasonable bounded synchronization error can be obtained. The clock synchronization error is importamt for multiprocess synchronization as well as for the correctness of the knowledge of time.

[1] If there are more than one possible configurations to execute this object, then there is a list $< c_{Id}[k] >$ for these possibilities.

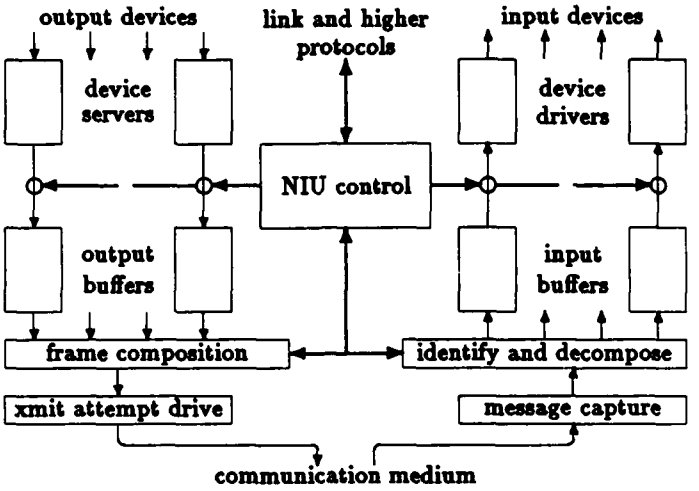


Figure 1: Network Interface Unit (NIU)

2 Architecture and Model

Our model is based on the observation that the errors in estimating communication delays can be decomposed into a deterministic part and a non-deterministic part. The deterministic part can be compensated for by an enhanced algorithm, and thereby this error term may be reduced to the non-deterministic part.

In order to derive a LAN timing error model, we first examine the LAN architecture as defined by hierarchical models ([17]). Both the OSI seven layer model, and the IEEE 802 standards, are built on top of a physical layer. The second layer in the OSI model is the link layer, which is represented in the IEEE model by two layers, the medium access control (MAC) and the logical link control (LLC). All these layers constitute the the local network (LN) protocols. On top of the local network layers, the higher layers provide a modular and flexible design environment, but they also introduce the delay and timing control problems. The architectural issues of these layers are not within the scope of this paper and for a further review of these issues refer to [17]; the delays and timing control issues are discussed here in details.

2.1 The Network Interface

Each of the nodes in the system is connected to the communication network through a network interface (NIU). The network interface interconnects three items: the logical link

protocol that is executed on the host, the devices to which (or from which) the data should be delivered (or got) and the communication medium. A general network interface unit is described in Figure 1.

The network interface contains servers and drivers. The device servers are used to *get* data from the output devices, while the drivers are used to *put* data to the input devices. We denote the time required for these actions t_{get} and t_{put}, respectively.

The data segments that are transferred between the devices and the communication medium are ***buffered*** in input and output buffers. The buffering is controlled by the NIU-control, allowing buffer filling at the protocol convenience, for various reasons as acknowledgements, flow control, etc. We denote the time a message "waits" at the buffers as t_{buf-in} and $t_{buf-out}$, respectively.

The NIU also *composes* the frames to be sent, by dividing a long message into fragments and appending the proper head/tail to the message. The time required for this action is denoted as t_{comp}. Similarly, when a message is received it must be identified ("Is it for me?") and *decomposed* by removing the head/tail. This time is denoted by t_{decomp}. Both t_{comp} and t_{decomp} contain CRC treatment (generation or check respectively).

The transmission of a message involves acquiring control on the communication medium. In a CSMA/CD system, *attempts* to transmit may result in collisions, followed by retransmissions according to the persistency of the unit. In a token network, the unit *waits* for a free token before attempting transmission. In some other architectures the unit waits for its time-slot to transmit. We denote the time that elapses from the moment a message is composed to the moment its successful transmission starts as $t_{attempt}$.

Once a successful message transmission starts, it takes t_{xmit} time units to transmit it. After a propagation delay of $t_{propagate}$ it reaches its receiving NIU. The *capture* mechanism receives the message, an action that consumes $t_{receive}$.

The economical benefits in serving a number of devices with one NIU raise some timing uncertainties. The state of NIU-control is not known at the time a message arrives, either from a device or from the communication medium. Hence, it is difficult to predict how long its response time is going to be. Furthermore, the fact that the logical link protocol is not synchronized with NIU-control introduces additional time uncertainties :

- Upon sending: from initiation to xmit/attempt.
- Upon receipt: from capture to link protocol interruption.

We denote the time from identification ("It is for me!") to interruption of the logical link protocol by t_{signal}. This time can also be expressed as $t_{buf-in} + t_{put}$, in cases data has to be delivered to an input device. But in cases a message is purely a control function, the NIU-control can inform the Link protocol in a shorter way, as can be observed in Figure 1.

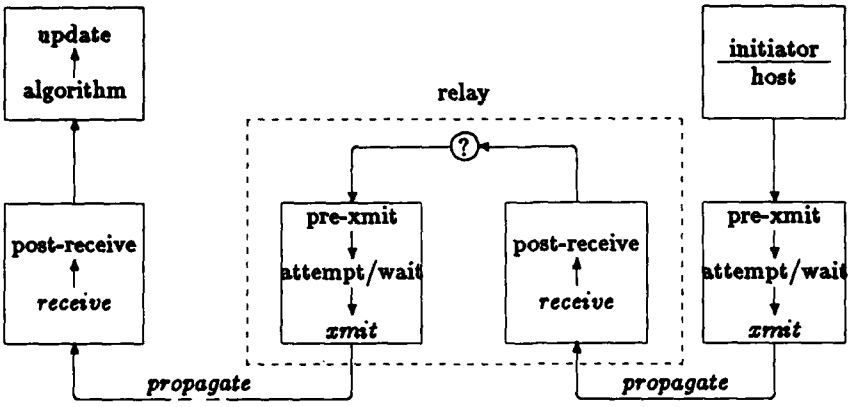


Figure 2: Delay model for a LAN

2.2 Communication Elements and Timing Uncertainties

A general description of the actions that produce delay in a local area network is given in Figure 2. The delay is generated and accumulated from the moment a host initiates a specific activity (e.g., reading a clock) to the moment a remote variable is updated (e.g., clock difference). In the general case, we assume there can be an indirect connection between the two communicants. In such a case relays are needed. For $n-1$ relays, n propagation delays are accumulated during a message transfer. We describe these delay contributors according to the layered architecture of a local area network. In order to simplify the description, we start with a restriction of an homogeneous implementation, i.e., all the NIU's and protocols are the same in all the sites and all the communication connections are the same. Later, this restriction is released.

2.2.1 Physical layer:

The delay from *xmit* to *propagate* to *receive* depends on properties of the physical layer:

- bandwidth,
- topology,
- medium technology.

The above, with the frame size, produce a highly predictable delay. As shown above, this delay can be parameterized, and expressed as

$$t_{xmit} + t_{propagate} + t_{receive}.$$

2.2.2 MAC layer:

The MAC activities consist of the the following:

- The attempt/wait phase in the transmission part.
- The frame composition in the pre-xmit phase.
- The frame identification and decomposition in the post-receive phase.

As shown above, this delay can be parameterized[2], and expressed as

$$t_{attempt} + t_{comp} + t_{decomp}.$$

2.2.3 Link layer:

The link layer initiates sending a message from a device, or a data delivery to a device. Its activities can be started (in case some conditions are satisfied, as discussed below) t_{signal} after NIU-control identifies a particular message arrived. The activities consist of the following:

- Moving data from the device to the NIU.
- Filling the output buffer.
- Filling the input buffer.
- Moving data from the NIU to the device.

As shown above, this delay can be parameterized, and expressed as

$$t_{get} + t_{buf-out} + t_{buf-in} + t_{put}.$$

Additional time is required for composition of an outgoing frame at that level, as well as for decomposition of an arriving frame. Similarly, if buffering is used at that level, additional delay may be introduced.

[2] We assume that although containing non-deterministic components, the MAC and physical layers' contributions to the delays are bounded. This assumption implies that *some* LAN protocols (e.g., Ethernet) are not to be used in real-time systems, because there is no bound for their delays.

2.3 A Communication Delay Model

For a specific implementation of a network, we merge the delays due to composition, decomposition and buffering of all the levels as if one level of the hierarchy generates them. Yet, in order to complete our model we need to account for another delay, the one introduced by the application program. We denote it as $t_{algorithm}$.

To summarize our delay model, let us assume the following:

- One application initiates transmission to another.
- There are $n-1$ communication relays between the source and the destination, as described in Figure 2.
- No recovery mechanism retransmits a lost message without a proper update of its content, since messages contain timestamps.
- Contention effects are contained within $t_{attempt}$.

We can derive a bound for the total message delay μ_i^j, and sum the maximal value of the contributors:

$$t_{algorithm} + n \times (t_{pre-xmit} + t_{attempt} + t_{xmit} + t_{propagate} + t_{receive} + t_{post-receive})$$

where:

$$t_{pre-xmit} = t_{get} + t_{buf-out} + t_{comp}$$

$$t_{post-receive} = t_{decomp} + t_{buf-in} + t_{put}.$$

This bound is too large for any real-time system and we examine an alternative approach, in which a large portion of the above bound is known a priori and compensated for.

The restriction on having identical elements can be removed, and the delay μ_i^j of a message from i to j through the relays r_1 to r_{n-1} can be written as the following sum (instead of the above multiplication)[3]

$$\mu_i^j = t_{comp}[i] + t_{attempt}[i] + t_{xmit}[i] + t_{propagate}[j] + t_{receive}[j] + t_{decomp}[j] + t_{signal}[j] + t_{algorithm}[j] +$$
$$+ \sum_{k=r_1}^{r_{n-1}} (t_{comp}[k] + t_{attempt}[k] + t_{xmit}[k] + t_{propagate}[k] + t_{receive}[k] + t_{decomp}[k] + t_{signal}[k])$$

The adequacy of LAN architecture for real-time applications has been examined before in a qualitative way ([11]). Our model, as expressed above, provides a quantitative way

[3]The indices relate each parameter to a participant node in the network. The index i stands for the source, the index j for the destination and k for the relays. In the rest of the document we omit the indices for better readability.

of expressing the delay of a message delivery in a LAN environment. The above model is general and may be applied to any local area network topology or protocol. The elements in the above sum equation contain, in general, a deterministic part and a non-deterministic part in each of them. However, each of these deterministic parts of the elements can be parameterized for a specific system configuration, and be used later in compensation procedures. We show the principle of such a compensation regarding clock synchronization algorithms, but first we introduce the principles of the algorithms themselves.

3 Time Synchronization

In a distributed system, different processing nodes that acquire the knowledge of time from different clocks need to synchronize these clocks. There are two different synchronization requirements, each of which originates in a system requirement. On one hand, clocks should be synchronized with respect to each other to allow multiprocess synchronization. In other words, the times acquired at any particular instance at distinct nodes should be close to each other. On the other hand, clock should be synchronized with respect to real-time, to allow correctness and accuracy of the knowledge of time. In real-time systems, where decisions are made in accordance with time constraints (see section 1), meeting the above two requirements are of crucial importance.

3.1 Clocks

We can divide the real-time systems into two groups:

1. Systems in which a computation is to meet a deadline, yet the computation itself does not require the knowledge of the Newtonian time [4].

2. System in which a deadline is to be met, and the computation uses the Newtonian time as well. An example of such a system is an inertial navigation system, that computes place coordinates by applying a double integration over time to the measured acceleration.

The second group generally imposes more stringent restrictions on the continuous correctness of the knowledge of the time than does the first group. Therefore, the second group is the one examined in this paper.

The source of this knowledge of the time is referred to as a *clock*. It is assumed that each site has an access to a clock, which it can read.

[4] We shall not consider aspects of non-Newtonian time.

3.1.1 Definitions

- Let $C_i(t)$ denote a function that maps real time to clock "i" time.
- A *standard* clock "i" is one with $\forall t : C_i(t) = t$.
- A clock "i" is *correct* at time t_0 if $C_i(t_0) = t_0$.
- A clock "i" is *accurate* at time t_0 if the first derivative of $C_i(t)$ is 1 sec/sec at t_0.

3.1.2 Clock Synchronization

Clock synchronization is used for ordering purposes ([4]), as well as for enhancement of the knowledge of the time ([7]). We consider such a synchronization as a clock update:

$$C_i(t) \leftarrow F(C_{i1}(t_{i1}), C_{i2}(t_{i2}), .., C_{ik}(t_{ik}))$$

where the function F depends on the algorithm we use.

3.2 Types of Clock Systems

We divide the clock systems into three categories: central dictatoric systems, central democratic systems and distributed systems ([1]). We now introduce these categories briefly; the latter two are examined in detail in Appendix A.

3.2.1 Central Dictatoric

This category is characterized by the following properties:

- One accurate clock feeds the whole system. The existence of other clocks in the system is "ignored" as long as no failure is detected in the central controller.
- Redundancy of the central controller is used for fault tolerance.
- The method is accurate (within nanoseconds to milliseconds, depending on the central clock) and expensive.
- This method needs a special purpose hardware integration into the processor, such that the central controller sets this hardware to the proper value, and the site can read it.
- The communication cost of this category is very low: only one message is required for each site synchronization. In a broadcasting environment, one message is required for a group of sites synchronization.

An example of this category is the GPS (Global Position System) that uses four broadcasting satellites, and achieves a clock correctness of nano-seconds.

We shall not consider this group further, because its implementation does not interact with any network implementation or network issue.

3.2.2 Central Democratic

This category involves distinguishing between two types of nodes. The following properties characterize this category:

- A nominated master clock polls slave clocks.
- Clock differences are measured and slaves are corrected.
- In case the master clock fails, an election of a new master (from the slaves) is initiated (e.g., [8]).
- Transmission times and delays are estimated, since they affect significantly the clock difference measured.

Examples of this category are Tempo ([2,3]), Etempo ([10,1]), DCNet ([12]).

Performance

It is shown in the appendix (see A.1) that the estimate of the clock skew ξ_j is obtained by

$$\hat{\xi}_j = \frac{d_1 - d_2}{2} = \xi_j + \frac{1}{2}(\mu_i^j - \mu_j^i) - \frac{1}{2}(E_j^1 - E_j^2)$$

where d_1 and d_2 are the measured differences between the clock of the master and the clock of the slave at the first and second synchronization messages, μ_i^j is the elapsed time of a message travelling from node i to node j and E_j^k $(k = 1, 2)$ are random noise components.

Note that if $\mu_i^j \approx \mu_j^i$, the error term due to communication is reduced. In addition, in cases E_j^k is known to be symmetrically distributed with a zero mean, a number of polls can be used for averaging the result ([2,1]), thereby reducing the error in the estimate.

$$\hat{\xi}_j = \xi_j + \frac{1}{2}(\overline{\mu_i^j} - \overline{\mu_j^i}) - \frac{1}{2}(\overline{E_j^1} - \overline{E_j^2}).$$

The communication cost of each slave update is 2 messages for a poll and one for the update. Therefore, n processors updated after p polls require $(2p + 1) \times n$ messages.

The error E_j^k is significantly influenced by the granularity of the clocks. The result of the averaging procedure $\frac{1}{2}(\overline{E_j^1} - \overline{E_j^2})$ is that granularity is dominated by the "worst" participant.

The assumptions that $\mu_i^j \approx \mu_j^i$ and $\overline{\mu}_i^j \approx \overline{\mu}_j^i$ are not realistic. A better knowledge of the communication times μ_i^j and μ_j^i, along with a reduction of these times, significantly improves the estimation of the clock skew $\hat{\xi}_j$. We show later how it can be done.

In order to maintain a bounded synchronization error between updates, consider the following model:

- the clock was synchronized at $t = \rho_j$ with a synchronization error of ϵ_j,
- ϵ_j is the error of the above algorithm

$$\frac{1}{2}(\overline{\mu}_i^j - \overline{\mu}_j^i) - \frac{1}{2}(\overline{E_j^1} - \overline{E_j^2}),$$

- the clock drift rate is δ_j, and
- the above algorithm is executed at least every τ time units.

Hence, the maximal clock difference is bounded by ([3])

$$|2\tau \times \max_j(\delta_j)| + |4 \times \max_j(\epsilon_j)|.$$

The frequency of updates (τ^{-1}) therefore controls the bounds on synchronization correctness between updates.

3.2.3 Distributed

Various examples of this category have been published. The first and most fundamental is by Lamport [4]. Later enhanced approaches include [6,5,7,9].

A distributed clock system can be characterized as follows.

- All the sites are homogeneous, each runs the same algorithm.
- Each site updates its own clock after receiving the time from other clocks and after estimating their correctness.
- The fault tolerance is based on communication. In case a site fails, the other sites are not affected. They only need to detect the failure and ignore that site.
- Generally, an enormous communication traffic is involved in this category, especially where robustness to presence of malicious faults is required ([5,9]).

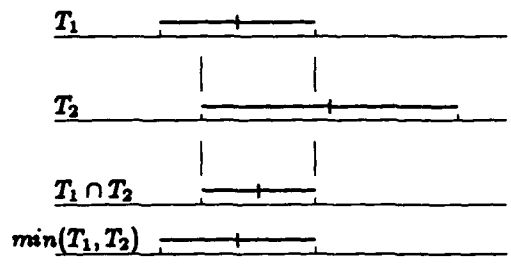


Figure 3: Synchronization algorithms by Marzullo and Owicki, 1985.

Performance

Two algorithms are introduced in [7]: the minimize-maximum-error algorithm and the intersection algorithm. The two are discussed in Appendix A and compared in Figure 3.

The minimize-maximum-error algorithm produces an error that can be expressed ([7]) as:

$$|C_i(t) - C_j(t)| < 2E_M(t) + 2\mu + (\delta_i + \delta_j)(\tau + 2\mu)$$

where E_M is the smallest error interval of the participants and μ is a bound for the *communication* time. The intersection of time intervals results in an error ([7]), when no inconsistencies occur :

$$|C_i(t) - C_j(t)| < \mu + (\delta_i + \delta_j)\tau.$$

The intersection algorithm is superior in the accuracy performance, as can be observed both from the above equations and from Figure 3. However, it is less robust: it can ignore all the responses due to one erroneous response. Therefore, although the synchronizing system does not fail upon an erroneous response, its fault tolerance performance is poor, since the algorithm is not executed.

Both algorithms require a good knowledge of the communication time. It is used for updating the intervals and for examining the the responses. Furthermore, even the fundamental ordering algorithm ([4]) requires this knowledge. In most cases a good estimate is sufficient, but precautions should be taken in order to have sufficient confidence in the estimate. This issue is addressed in the next section.

The assumption of having a good estimate for the communication time is different from what we had in the master-slave approach. There, the assumption was $\mu_i^j \approx \mu_j^i$, which means we want both communication times to be the same.

4 Model Based Implementation

This section describes an example of possible implementations of a distributed and a central democratic clock synchronization algorithms. The major focus is the creation of an environment in which communication delays are predictable and can be compensated for. There are two possible environments for synchronized real-time clock systems to consider. In the first, time services (get-time-of-day, etc.), including the clock synchronization, are provided by the operating system. In the second, the application itself operates an independent synchronization algorithm, but the operating system supports precedence requests and urgency modes. The first is preferrable, since the application has only to deal with the operating system. All the protocols relations in this case are supported by the operating system, without any application intervention.

There are four participants in the clock synchronization algorithm for a real-time system, excluding the clocks: the application program, the operating system, the high-level communication protocols and the local network. In order to achieve high accuracy, they should all be "real-time oriented", in the sense of being timely correct. But this is rarely the case. There are no high-level communication protocols which are targeted for real-time applications. Therefore, we assume that real-time efficiency and determinism exist at least at the application program and at the operating system. In addition, we assume that the local network allows imposing at least precedence relation between messages, if circuit switching is not possible.

There is no use in implementing a real-time system with clock synchronization algorithm, in cases where the bounds on the communication durations are larger than the correctness requirement. With the above assumptions, TCP/IP serve as good examples of high level protocols with which one has to implement this type of systems. A summary of these protocols is provided in appendix B, including their precedence control and their timestamp mechanism.

Synchronization can be activated at lower level protocols (e.g., ICMP instead of TCP), with no preliminary procedures required from the application, as those shown below. Yet, in most cases this service is not supported, and the result is seen in increased uncertainties, due to lack of control over the lower levels. We now discuss both cases, recalling that we assume precedence and urgency are supported by the local network (NIU, MAC protocol and logical link protocol) and by the TCP/IP package.

4.1 Application Level Synchronization

The application program implements some steps in which the synchronization algorithm is activated. First, a proper link is established. Then, *polls* by the relevant clock synchro-

nization algorithm are executed for each clock (see section 3.1). For each poll an urgency mode is set, and the poll is then carried out. Finally, the link is released.

Connection Establishment

To establish a connection in TCP, an OPEN command is used. Upon starting the connection, the precedence field should be set to "low delay". The TCP options field may be used [5] to impose activation of the ICMP, but other ways are also possible. This step should result in a connection in which all the participants (source, destination and relays) impose low delay for message transfers. Furthermore, the IP modules should apply frame formats with the timestamp options field (see Figure 9) for this logical channel. The options field size is to be set to the number of relays in this connection, allocating a space for timestamps to be collected. As the message travels, each of the relays appends its timestamp and identifier to the message. In case there are more participants, the OVFLW counter captures them, and the field may be updated for latter polls. The purpose of this restriction is to provide the routing information needed for the communication duration estimate.

Pre-poll Setting

Before each poll, an urgency state is to be imposed on the participants in the connection. A preliminary command is sent by using the TCP flags (PUSH and URG), and setting the urgent pointer to the $current\_sequence\_number + p$, where p is the number of polls to be carried out. The result of this step is a buffer-cleaning throughout the connection, and an urgency state that holds the receiver as long as the polls are not complete. Hence, the uncertainties due to pre-xmit and post-receive are significantly reduced. In other words,

$$\Delta t_{get} + t_{buf-out} + t_{buf-in} + \Delta t_{put} \rightarrow 0.$$

With the above assumptions, the NIU can be assumed to provide a predictable t_{signal}. For $n-1$ relays and the receiver it is accumulated to

$$\sum_{1}^{n} t_{signal} = \sum_{1}^{n} (t_{get} + t_{put}).$$

Interrupts that might influence the algorithm execution time should be disabled ([2]), and thus

$$\Delta t_{algorithm} \rightarrow 0.$$

Therefore, estimating the communication duration

$$\mu = t_{algorithm} + \sum_{1}^{n} (t_{comp} + t_{attempt} + t_{xmit} + t_{propagate} + t_{receive} + t_{decomp} + t_{signal}) + \Delta\mu$$

[5] One needs access to the TCP-IP connection in order to do it. So far only three options are defined for TCP options field, and no TCP-ICMP connection is defined at all [14,15].

or in other words

$$\mu = \hat{\mu} + \Delta\mu.$$

If $\hat{\mu} \approx \mu$,

$$\Delta\mu \approx \sum_{1}^{n} t_{attempt}.$$

All the other terms are predictable and compensated for, using the routing information that is contained within the timestamp option field.

4.2 Enhanced Algorithms

The above result, of a parameterized deterministic communication, may be used to enhance the performance of the synchronization algorithms in Appendix A, by compensating for the deterministic part of the communication duration. Two examples follow.

4.2.1 The Algorithm: Master–Slave

do
  for $k = 1$ to p:
    $T_1 \leftarrow C_i(now)$;
    Send(T_1) ;
    Receive($T_2, T_3, \hat{\mu_i^j}$) ;
    Read-Status(T_4, n) ;
      /* n is the number of relays */
    estimate μ_j^i from n and timestamps using the above equation for $\hat{\mu}$;
      /* $\Delta t_{propagate} + \Delta t_{xmit} + \Delta t_{receive} \rightarrow 0$ */
    Compute C_j skew $\Delta_j[k]$;
      /* $d_1^j = T_2 - T_1 - \hat{\mu_i^j}$ */
      /* $d_1^j = T_4 - T_3 - \hat{\mu_j^i}$ */
      /* $\Delta_j[k] \leftarrow \frac{1}{2}(d_1^j + d_2^j)$ */
  Compute C_j skew Δ_j from $\Delta_j[1..p]$;
endo

Hence,

$$\Delta\mu \rightarrow \sum_{1}^{n} t_{attempt}.$$

4.2.2 The Algorithm: Distributed

do
  for $k = 1$ to p:
    Send(Request) ;
    Receive($T[k], E[k]$) ;
    Read-Status($C_i[k], n[k]$) ;
      /* $n[k]$ is the number of relays from clock k */
    estimate μ_k^i from n and timestamps using the above equation for $\hat{\mu}$;
      /*$\Delta t_{propagate} + \Delta t_{xmit} + \Delta t_{receive} \rightarrow 0$*/
  compute C_i error in Algorithm intersection (see A.5.3)
endo

Hence,

$$\Delta\mu_j^i \rightarrow \sum_1^n t_{attempt}.$$

4.3 Operating System Synchronization

In the application level synchronization we have assumed that interrupts can be disabled, ICMP messages can be used, and precedence and urgency control options are available. In many cases, due to security reasons and others, this is not feasible. The operating system support is then the only possibility for a clock synchronization with a reasonable accuracy.

In case of a distributed real-time operating system, the time service involves replicas of the operating system, located at different sites. In case of a simple network, each site has its own real-time operating system, and each pair of sites may synchronize with others. In both cases, the access to ICMP for executing the proper synchronization is direct, and there is no need for TCP involvement. Thus, one receives even lower t_{get} and t_{signal}, using the direct access of the local link protocol to the NIU-control.

5 Conclusion

In this paper we model the behavior of a local area network such that real-time characteristics of its services can be analyzed. In particular, the clock synchronization service was examined, being the most crucially dependent on communication delays. A model was built, and communication time between two sites that are n links apart from each other was formulated.

The approach, the model and the results presented in this paper are useful in designing real-time systems with LAN communication environment. The applicability of our approach is shown to be adequate for designing time-servers for real-time systems. Not all the aspects of time-service are dealt with here (e.g., fault tolerance and "on demand" timestamping). However, the determinism of clock synchronization algorithms is orthogonal to these other aspects, and they all benefit from its enhancement.

It was shown that under precedence and urgency modes, and a very careful implementation of LAN protocols, the uncertainties of the communication time can be reduced. The deterministic part of the delay in a message delivery can be derived from routing information which is appended to the messages and can be compensated for. The compensation techniques, in general, depend on the specific application. For example, topology of the network and the protocols play a major role in the implemented technique. We therefore do not include a detailed technique in this paper. However, both distributed and central-democratic algorithms result in enhanced correctness and accuracy using the approach devised here.

References

[1] Gora W., Herzog U. and Tripathi S., *Clock Synchronization on the Factory Floor*, Proc. of Workshop on Factory Communication, NBS, March 1987.

[2] Gusella R. and Zatti S., *TEMPO - Time Services for the Berkeley Local Network*, Report No. UCB-CSD83-163, Computer Science Division, University of California, Berkeley CA, December 1983.

[3] Gusella R. and Zatti S., *The Accuracy of Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD*, technical report, Computer Science Division, University of California, Berkeley CA, December 1986.

[4] Lamport L., *Time, Clocks and Ordering of Events in a Distributed System*, Communications of the ACM, Vol 21 No 7 pp 558-565, July 1978.

[5] Lamport L. and Melliar-Smith P. M., *Synchronizing Clocks in the Presence of Faults*, Journal of the ACM, Vol 32 No 1 pp 52-78, January 1985.

[6] Lundelius J. and Lynch N., *A New Fault Tolerant Algorithm for Clock Synchronization*, M. I. T. Laboratory of Computer Science, Cambridge MA, June 1984.

[7] Marzullo K. and Owicki S., *Maintaining the Time in a Distributed System*, ACM Operating Systems Review, Vol 19 No 3 pp 44-54, July 1985.

[8] Ricart G., Agrawala A., *An Optimal Algorithm for Mutual Exclusion in Computer Network*, Communication of the ACM, Vol 23 No 1 pp 9-17, Jan 1981.

[9] Shin K. and Ramanathan P., *Clock Synchronization of a Large Multiprocessor System in the Presence of Malicious Faults*, IEEE Trans on Computers, Vol C-36 No 1 pp 2-12, January 1987.

[10] Tripathi S. and Chang S., *A Clock Synchronization Algorithm for Hierarchical LANs – Implementation and Measurements*, Technical Report TR-86-48, Systems Research Center, University of Maryland, College Park MD, 1986.

[11] Le Lann G., *Issues in Fault-Tolerant Real-Time Local Area Network*, Proceedings of the Fifth Symposium on Reliability in Distributed Software and Database Systems, pp 28-32, Los Angeles, CA, January 1986.

[12] Mills D. L., *DCNET Internet Clock Service*, RFC-778, Defense Advanced Research Projects Agency, Information Processing Techniques Office, April 1981.

[13] Postel J., *Internet Protocol*, RFC-791, Defense Advanced Research Projects Agency, Information Processing Techniques Office, Sept. 1981.

[14] Postel J., *Internet Control Message Protocol*, RFC-792, Defense Advanced Research Projects Agency, Information Processing Techniques Office, Sept. 1981.

[15] Postel J., *Transmission Control Protocol*, RFC-793, Defense Advanced Research Projects Agency, Information Processing Techniques Office, Sept. 1981.

[16] Stallings W., *The DOD Communication Protocol Standards*, SIGNAL magazine, Vol 40 No 8 pp 29-34, April 1986

[17] Stallings W., *Local Networks*, second edition, Macmillan Publishing Company, New York, New York 1987.

A Clock Synchronization Algorithms

A.1 Master – Slave Algorithms

A master clock initiates a clock synchronization procedure, at a time T_1. It reads its current clock value, which is $C_i(T_1)$, having an error e_1. It sends this clock value to a slave. The message travels for a period of μ_i^j, and is received by the slave at T_2. At that time the slave clock has the value of $C_j(T_2)$ and an error e_2. The slave can now compute the difference

$$d_1 = C_j(T_2) - C_i(T_1).$$

Comparing the receiving time to the sending time

$$C_i(T_1) + e_1 + \mu_i^j = C_j(T_2) + e_2$$

yields

$$C_j(T_2) - C_i(T_1) = \mu_i^j + (e_1 - e_2).$$

If we model the error difference such that ξ_j is the clock j skew and E_j^k $(k = 1, 2, ..)$ is a noise,

$$d_1 = \mu_i^j + \xi_j - E_j^1.$$

Now, the process can be repeated, but in the opposite direction. The slave reads its current clock value at a time T_3, which is $C_j(T_3)$ having an error e_3. It sends this clock value to the master, attaching d_1 to it. The message travels for a period of μ_j^i, and received by the master at T_4. At that time the master clock has the value of $C_i(T_4)$ and an error e_4. The master can now compute the difference

$$d_2 = C_i(T_4) - C_j(T_3).$$

Comparing the receiving time to the sending time

$$C_j(T_3) + e_3 + \mu_j^i = C_i(T_4) + e_4$$

yields

$$C_i(T_4) - C_j(T_3) = \mu_j^i - (e_4 - e_3)$$

and as above

$$d_2 = \mu_j^i - \xi_j - E_j^2.$$

From the above we receive

$$\frac{d1 - d2}{2} = \xi_j + \frac{1}{2}(\mu_i^j - \mu_j^i) - \frac{1}{2}(E_j^1 - E_j^2).$$

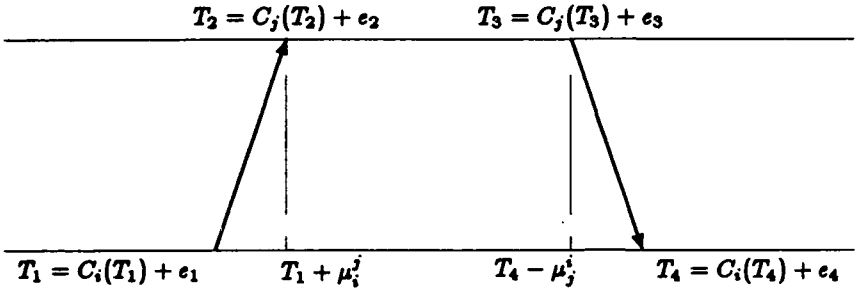


Figure 4: Master – Slave synchronisation principle.

Assuming $\mu_i^j \approx \mu_j^i$ reduces this term significantly, and assuming E_j^k to be symmetrically distributed with a zero mean can be used in performing a number of polls and averaging the results ([2,1]).

A.2 Tempo : a Master–Slave example.

Tempo is a clock synchronization algorithm that was developed for the distributed Berkeley Unix ([2]). The algorithm was extended to hierarchical networks (ETempo [10]). Similar algorithm is provided by the DCNET ([12]), and supported by the ICMP protocol messages for time stamp and time stamp reply ([14]).

A.2.1 Algorithm for Master P_i:

- *Upon Initiation of Clock Skew Measurement*::

  do
  $T_A \leftarrow C_i(now)$;
  $\forall j \neq i$: Send(T_A) to j
  endo

- *Upon Receiving* (T_B, d_1^j) *from Slave* j ::

  do
  $d_2^j \leftarrow C_i(now) - T_B$;
  /* $d_2^j = \mu_j^i - \xi_j - E_j^2$ */
  $\Delta_j \leftarrow \frac{1}{2}(d_1^j + d_2^j)$

$$/* = \xi_j + \tfrac{1}{2}(\mu_j^i + \mu_i^j) - \tfrac{1}{2}(E_j^1 - E_j^2)\ */$$

endo

- $\forall j \neq i$: *Upon Complete Receiving All Polls* ::

do

$\Delta \leftarrow \frac{1}{N}\sum_{k=1}^{N} \Delta_j[k]$;

$$/* = \overline{\xi_j} + \tfrac{1}{2}(\overline{\mu_i^j} - \overline{\mu_j^i}) - \tfrac{1}{2}(\overline{E_j^1} - \overline{E_j^2})\ */$$

Send(Δ) to j

endo

□

A.2.2 Algorithm for Slave P_j:

- *Upon Receiving* (T_A) *from Master* i ::

do

$d_1^j \leftarrow C_j(now) - T_A$;

$$/*\ d_1^j = \mu_i^j + \xi_j - E_j^1\ */$$

$T_B \leftarrow C_j(now)$;

Send(T_B, d_1^j) to i

endo

- *Upon Receiving* (Δ) *from Master* i ::

do

$C_i(t) \leftarrow C_i(t) + \Delta$

endo

□

A.3 Distributed Clock Algorithms

The distributed approach is more expensive than the master-slave approach, especially in the load it imposes on the communication network. Yet, its major advantage is the higher degree of fault tolerance it achieves, increasing the cost mainly in communication, rather than in special hardware. In the distributed approach all the clock owners use a uniform approach, whose characteristics are as follows.

- Polling the rest of the clocks, or a subset of the rest.
- Applying a specific algorithm to the responses of the poll.
- Updating the local clock accordingly.

Apart from the algorithms introduced below, there are many others that deal with fault tolerance enhancement ([6,5,9]). The algorithms we introduce below were chosen to represent the ideas of ordering and accuracy enhancement, and their dependency on network properties.

A.4 A Fundamental Ordering Approach

A basic ordering approach was introduced by L. Lamport in [4].
The **assumptions** taken are as follows:

- The clock *accuracy* is bounded by a drift rate δ

$$\forall t : |\frac{dC_i(t)}{dt} - 1| < \delta_i \ll 1.$$

- The communication graph is closely connected with a diameter d.
- The network imposes an unpredicted message delay, D, such that $\mu < D < \eta$, where μ and η are the lower and upper bounds on D respectively.

The **algorithm** adopted in [4]:

- Each process with a clock sends messages to the others at least every τ seconds. Each message includes its timestamp T_m.
- Upon reception of an external T_m, the receiver sets its clock

$$C_i(t) \leftarrow max(C_i(t), T_m + \mu).$$

□

The correctness of each clock synchronization, as achieved in this algorithm is:

$$\forall i : \forall j : |C_i(t) - C_j(t)| < d(2\delta\tau + \eta)$$

for all t. The communication cost is $n \times (n-1)$ messages for one update of the whole network. Yet, this algorithm achieves only the ordering goal and results in updates according to the fastest clock in the system, which is not necessarily the most accurate one.

A.5 Time Intervals Approach

Two important distributed algorithms are introduced in [7], both taking advantage of the clock bounded incorrectness. Requiring the knowledge of this bound is a fairly easy restriction, since the information is provided in the manuals of the equipment in use. These algorithms have also a communication traffic of $O(n^2)$.

A.5.1 Assumptions and Error Model

Every clock owner i "knows" it is correct within the interval

$$[C_i(t) - E_i(t), C_i(t) + E_i(t)]$$

where $E_i(t)$ is a bound on i clock maximal error. The error interval is constructed from the following contributors:

- The error that comes into effect right on the clock *reset* time (ρ_i), as discretization and other constant errors (ε_i).
- The *delay* between the time this clock (i) is read until another clock (j) uses this readout for its update (μ_i^j).
- The *degradation* of time counting that develops between consecutive resets (δ_i).

A.5.2 Algorithm: Minimize maximum error for P_i

- *Upon receiving a time Request from $j \neq i$::*

  do
  $E_i(t) \leftarrow \varepsilon_i + (C_i(t) - \rho_i)\delta_i$;
  Send$(C_i(t), E_i(t))$ to j
  endo

- *At least once every τ time units* ::

do
$\forall j \neq i$: Request$(C_j(t), E_j(t))$;
for $j \neq i$ do begin
Receive$(C_j(t), E_j(t))$;
if $(C_j(t), E_j(t))$ is consistent with $(C_i(t), E_i(t))$
then if $E_j(t) + (1 + \delta_i)\mu_j^i \leq E_i(t)$
then begin /* update */
$C_i(t) \leftarrow C_j(t)$;
$\varepsilon_i \leftarrow E_j(t) + (1 + \delta_i)\mu_j^i$;
$\rho_i \leftarrow C_j(t)$
end
else *ignore it*
end
endo

□

A.5.3 Algorithm: Intersection of time intervals for P_i

- *Upon receiving a time Request from $j \neq i$*::

do
$E_i(t) \leftarrow \varepsilon_i + (C_i(t) - \rho_i)\delta_i$;
Send$(C_i(t), E_i(t))$ to j
endo

(exactly as in the previous algorithm).

- *At least once every τ time units* ::

do
$\forall j \neq i$:Request$(C_j(t), E_j(t))$;
$\forall j \neq i$: Receive$(C_j(t), E_j(t))$;
$\forall j \neq i$: $T_j(t) \leftarrow C_j(t) - E_j(t)) - C_i(t)$;
/* T represents the left boundary */
$\forall j \neq i$: $L_j(t) \leftarrow C_j(t) + E_j(t)) + (1 + \delta_i)\mu_j^i - C_i(t)$;
/* L represents the right boundary */

$\alpha \leftarrow max(T_j)$; $\beta \leftarrow min(L_j)$;
/* intersect all boundaries */
if ($\alpha < \beta$) /* consistent boundaries */
then begin
$\varepsilon_i \leftarrow \frac{1}{2}(\beta - \alpha)$;
$C_i(t) \leftarrow \frac{1}{2}(\beta + \alpha)$;
$\rho_i \leftarrow \frac{1}{2}(\beta + \alpha)$
end
else *ignore them all*
endo

□

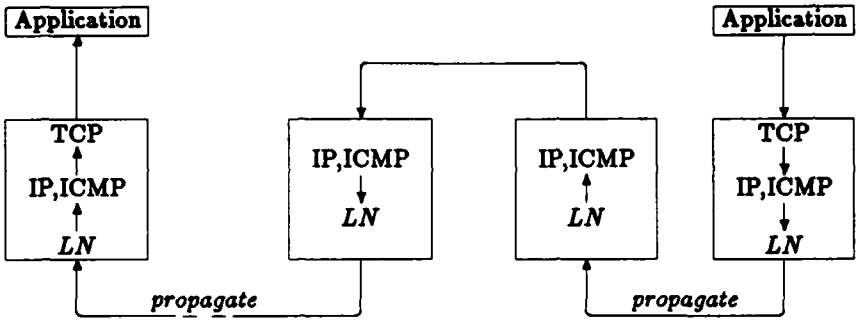


Figure 5: TCP – IP model

B Communication Protocols

B.1 Protocol Hierarchy Example

We demonstrate the environment for clock synchronization for real-time system, using the TCP ([15]) and IP ([13]) communication protocols. The way these two protocols are activated by the application levels can be seen in Figure 5. The result of these relations is a multilayer architecture, as the figure clearly illustrates.

The U.S. Department of Defense adopted these two protocol ([16]) as the layers between the local network layers (LN, i.e. physical $\longleftrightarrow$ medium access $\longleftrightarrow$ logical link) and the application layer. The Internet Protocol, IP MIL-STD 1777, provides the capability for end systems to communicate across one or more networks. The IP does not depend on the networks to be reliable, providing what is known as unreliable connectionless service. The Transmission Control Protocol, TCP MIL-STD 1778, provides a reliable end-to-end data transmit service, ensuring error free, in sequence, with no loss or duplication deliveries. In many ways it is equivalent to OSI layer-4 transport protocol.

The MIL-STD IP is further required to interconnect a set of networks (a *catenet* [16]). Hence, an IP module in a host or a gateway must be equipped with the information needed to make correct routing decisions. In addition, it must possess the knowledge to use the proper service access points (SAPs) of the local network (LN) layers.

The MIL-STD TCP provides its services to application layer programs as well as to protocols at higher level than his [6]. TCP numbers data segments it sends in a sequential manner. The transmitted segments are subsequently acknowledged by this number by the destination TCP module, hence providing means to reorder and detect lost messages. Con-

[6]Examples are the file transfer protocol, FTP MIL-STD 1780, and simple mail transfer protocol, SMTP MIL-STD 1781.

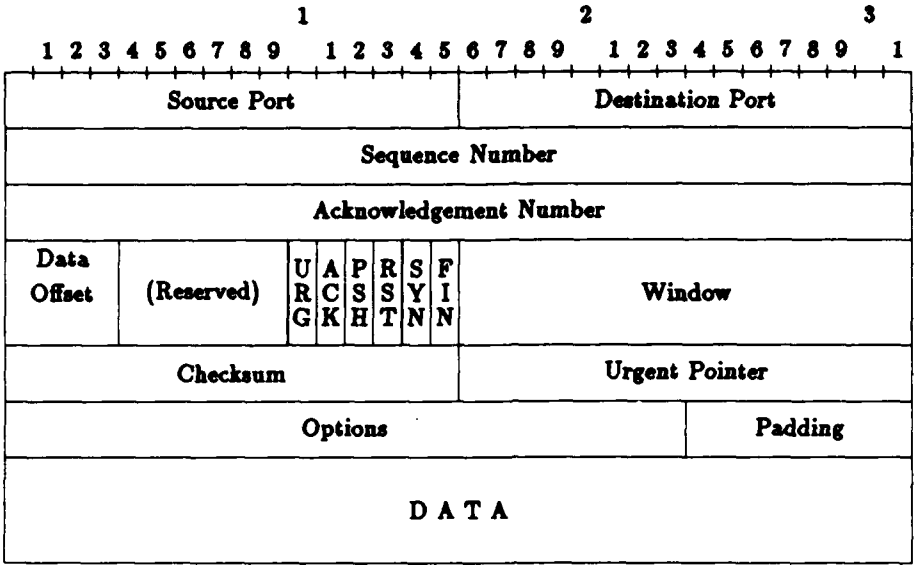


Figure 6: TCP Frame Format.

trols on the quality of transmission, the urgency and security are provided and described below.

In the next sections we describe the way in which the above requirements are achieved, by frame format definitions and by the interfaces to lower and higher layer levels.

B.1.1 TCP

A TCP frame format is described in Figure 6. Using this message structure, which encapsulates data and control, the TCP provides the user with SAPs (service access points [17]). We now describe the bidirectional interface.

User to TCP

The primitives and their parameters:

Open (local-port, foreign-socket, active/passive [,timeout] [,precedence] [,security] [,options]) $\Longrightarrow$ local-connection-Id.

Send (local-connection-Id, buffer-address, byte-count, PUSH-flag, URGent-flag [,timeout]).

Receive (local-connection-Id, buffer-address, byte-count) $\Longrightarrow$ byte-count, PUSH-flag, URGent-flag.

Close (local-connection-Id).

Status (local-connection-Id) $\Longrightarrow$ status data.

Abort (local-connection-Id).

TCP to User

Generation and transfer of asynchronous signals that are caused by events is based on the operating system services. The required ones are:

- Segment arrived.
- User timeout.
- Retransmission timeout.
- Time-wait timeout.

The information provided to the user by the TCP is:

- The local-connection-Id.
- Response string.
- The buffer-address.
- The byte-count.
- The PUSH-flag.
- The URGent-flag.

B.1.2 IP and ICMP

An IP frame format is described in Figure 7. Using this message structure, the IP provides the user with an interface (SAPs: service access points [17]). We now describe the bidirectional interface.

IP to upper levels

The primitive used and their parameters are:

Send (Source, Destination, Protocol, Type-of-Service, Buffer-pointer, Length-of-buffer, Identifier, Don't-Fragment, Options). $\Longrightarrow$ Result (OK/error).

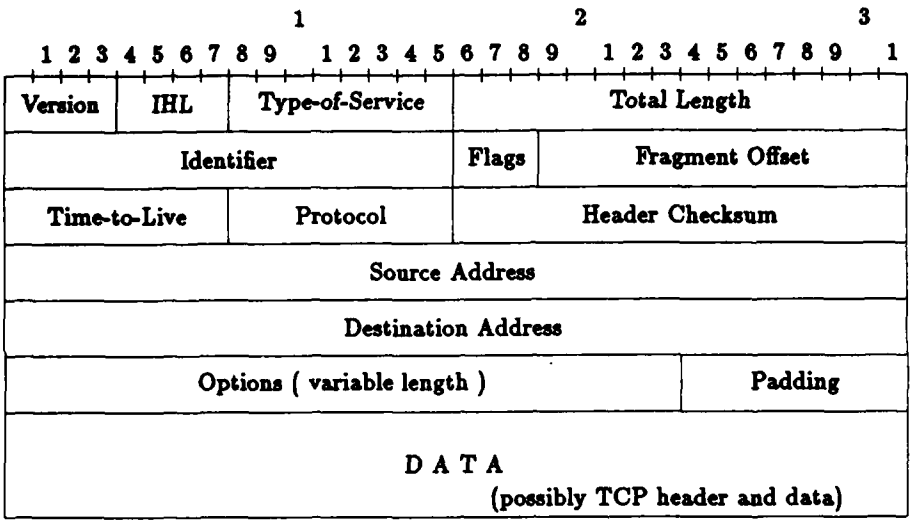


Figure 7: IP Frame Format.

Receive (Buffer-pointer, protocol). $\Longrightarrow$ Source, Destination, Type-of-Service, Length-of-buffer, Options, Result (OK/error).

TCP and IP Interface

In many cases (e.g the ARPA internetwork system [13,15]) the Type-of-Service and Time-to-Live fields are defined in a very loose way, unless user programs change it:

- Type-of-Service:
  1. precedence: routine,
  2. delay: normal,
  3. throughput: normal,
  4. reliability: normal,

  (i.e. 00000000).
- Time-to-Live: 1 minute (i.e. 00111100).

IP to lower levels

These lower levels are the NIU and MAC level functions, as described above.

ICMP messages

ICMP uses the IP frame format to transfer its messages. ICMP provides a set of messages to allow network control, exception handling, and error detection and handling. The messages are:

- Destination-unreachable.
- Time-exceeded.
- Parameter-problem.
- Source-quench.
- Redirect-datagram.
- Echo and echo-reply.
- Time-stamp and time-stamp-reply.
- Info-request and info-reply.

B.2 TCP Urgent Information Push

The objectives of an urgency mode in an asynchronous protocol, particularly in the TCP, are as follows.

- Allow a *sending user* to stimulate a *receiving user* to accept some urgent data.
- Permit a *receiving TCP* to indicate to a *receiving user* when all currently known urgent data has been received.

We examine the use of this mode and its effects, for an initiating user of a synchronization (e.g. master) to stimulate a receiving user (e.g. slave). In addition, a receiving TCP indicates the end of the poll to the receiving user.

B.2.1 TCP Variables that Enforce Urgency

Let *RCV.NXT* be a TCP variable representing the currently received sequence number. The following frame elements provide the means for urgent mode control:

- PSH bit: the segment contains data to be pushed through.
- URG bit: this urgent pointer is significant.
- *Urgent pointer*: points the end of urgent info within the data stream.

B.2.2 Urgent and Push Mechanisms

- **Push:**
  1. A sending TCP is allowed to collect data from a sending user (and send it later) until a PSH is signaled; then, it must send whatever it has.
  2. A receiving TCP is allowed to collect data before indicating to a receiving user until it sees a PSH; then, it must pass to the receiving user whatever it has, even if the buffer is not full.
- **Urgent:**
  1. When *Urgent pointer* is in advance of $RCV.NXT$, the receiving TCP informs the receiving user to go into an *urgent* mode.
  2. When $RCV.NXT$ catches up with *Urgent pointer*, the receiving TCP informs the receiving user to go into a *normal* mode.
  3. At least one data octet is required.
  4. PSH and URG: timely delivery of the urgent info to the destination process is enhanced.

B.2.3 Effects on Error Model

Applying the PUSH and URGent mechanisms to the model introduced in section 2 significantly reduces

$$t_{get} + t_{buf-out} + t_{buf-in} + t_{put},$$

thereby reducing t_{signal}. Furthermore, when an urgent mode is set by the initiator and the receiver applications, $t_{algorithm}$ can also be reduced significantly. The uncertainties remaining in these parameters can be controlled by the application, and allow compensation with high confidence levels.

B.3 Internet Protocol Precedence Control

B.3.1 Type-of-Service Control

The Type-of-Service field of the Internet Protocol is described in Figure 8. It is divided to the following:

- Precedence: routine, priority, immediate, ... internetwork control, network control.
- D: delay = low/high.

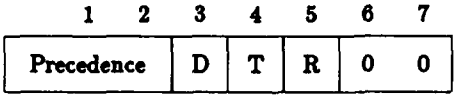


Figure 8: IP Type-of-Service Field

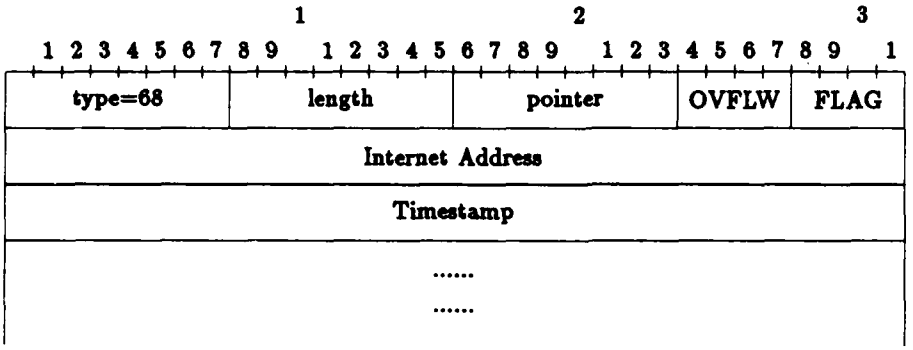


Figure 9: IP Options Field for Timestamping

- R: reliability = high/low.
- T: throughput = high/low.

B.3.2 Options field control

The Options field of the Internet Protocol, as defined for its time-stamp configuration, is described in Figure 9. It is controlled by the following:

- *pointer* > *length*: time-stamps area is full.
- OVFLW: a count of how many tried to add their time-stamp and didn't succeed.
- FLAG=0: time-stamp only.
- FLAG=1: each time-stamp preceded by its address.
- FLAG=3: time-stamp without addresses ordered as predefined.

These options can serve the algorithm one implements to collect one or more time stamps on the message route. Thus, communication load is reduced in some topologies (e.g. one token cycle for the whole net) and useful routing data is returned (e.g. number of hoops).

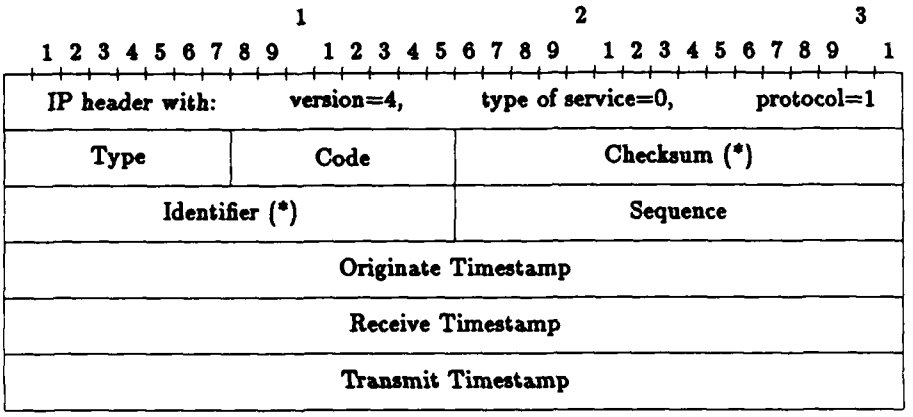


Figure 10: ICMP Timestamp Frame Format.

B.3.3 Effects on Error Model

The enforcement of low delay is not well defined in the specification of the Internet Protocol. The precedence control is also loosely defined. The releases allowed by the standardization committees do not guarantee a specific timing response. Still, using the proper (highest) precedence for a clock synchronization message, will result in a proper implementation in reducing the delays in the relays, although most of today's gateways simply ignore it.

B.4 ICMP Clock Synchronization Messages

ICMP provides a special message procedure to allow a master-slave clock synchronization, of the type described above in section A.1. To use this facility one uses the frame format described in Figure 10 using the following parameters.

- type=13, code=0: originator sends its time stamp.
- type=14, code=0: receiver replies.
- Identification and Sequence serve as session tags.

A similar service is provided by the DCNet ([12]), excluding the fields of Identifier and Checksum.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A185 475

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CS-TR-1892; UMIACS-TR-87-35	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION University of Maryland 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) Department of Computer Science University of Maryland College Park, MD 20742	7b. ADDRESS (City, State, and ZIP Code) 800 North Quincy Street Arlington, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-87-K-0124 N00014-87-K-0463
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) On Real-Time Systems and Local Area Networks	
12. PERSONAL AUTHOR(S) Shem-Tov Levi, Satish K. Tripathi	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) July 1987 15. PAGE COUNT 34
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB-GROUP	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The design and the implementation of a real-time communication network has to meet goals of fault-free and timely delivery of messages, as defined by the applications served by this network. The time constraints of the applications are projected as stringent constraints on the local knowledge of time. However, the local knowledge of time in a distributed system requires the use of the communication network. The accuracy and coherence of this knowledge depend on the way clocks are synchronized. Therefore, the performance of clock synchronization algorithms due to the communication network, is a good criterion for analyzing the adequacy of a particular network architecture to real-time application.

In this paper, we examine the adequacy of LAN architecture for real-time applications through a error estimates for these algorithms are derived. An efficient environment, in which the uncertainties in message communication elapsed time is significantly reduced, is then described for a layered protocol architecture. In addition, it is

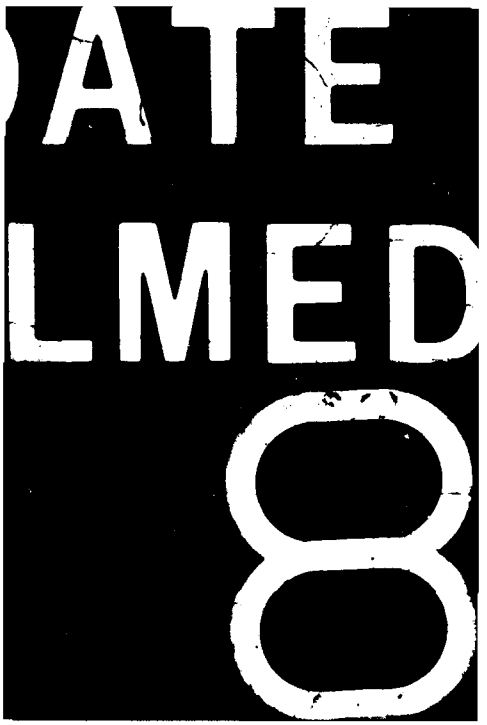
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

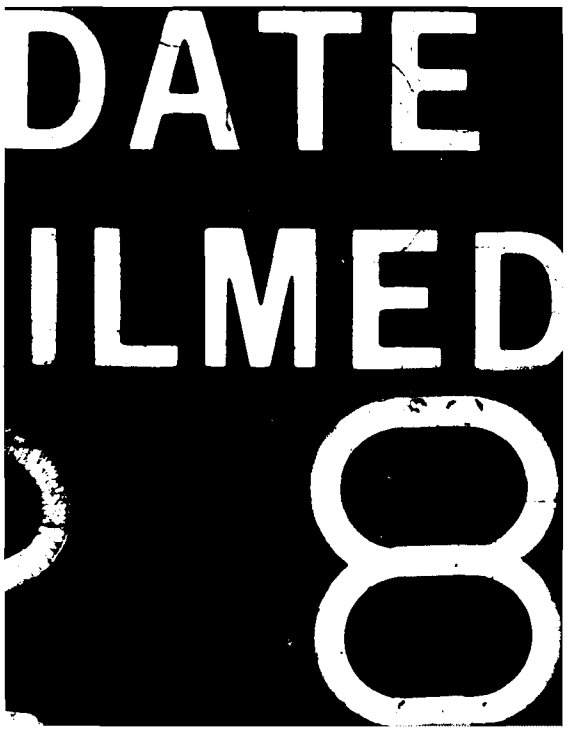
DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

shown that predictable communication elapsed time can be compensated for in synchronization algorithms, significantly reducing the errors of these algorithms. As an example of hierarchical protocols for this environment, we have chosen to use TCP, IP, and ICMP.


ATE
LMED
8


DATE
ILMED
8